

Solving $Ax=b$ using the Lanczos Algorithm with Selective Orthogonalization.

Horst Simon¹ and Beresford N. Parlett².

ABSTRACT

This report is a description and a user guide for the FORTRAN program LANSO. LANSO computes an approximate solution vector for the system of linear equations $Ax = b$, using the Lanczos tridiagonalization algorithm with selective orthogonalization. Here A is a symmetric, real $n \times n$ matrix and b is a given n -vector. Since this idea is rather new the theoretical background for LANSO is discussed briefly. The structure and the subroutines of LANSO are discussed in detail, and a sample run of LANSO is given. The main applications are to problems where the matrix A is large and sparse, the cost of the matrix vector product dominates other costs, and/or the matrix is not explicitly available.

1. Introduction.

In many applications one encounters the intermediate task of computing a solution vector x to the system of linear equations

$$Ax = b, \quad (1.1)$$

where A is a symmetric $n \times n$ matrix and b is an n -vector. If A is large and sparse, there is an elegant way to exploit the sparsity by employing A only as a linear operator which computes Av for any given vector v . There are several methods known, which produce an approximate solution vector based only on repeated computation of matrix vector products, e.g., the method of conjugate gradients (CG) by Hestenes and Stiefel [1], the algorithm SYMMLQ by Paige and Saunders [4], Lanczos' method of minimized iterations (LAN), and the Lanczos algorithm with selective orthogonalization (LANSO) by Parlett [6].

All these methods have several attractive features in common. There are no special properties needed for A , no acceleration parameters have to be estimated, and the storage requirements are only a couple of n -vectors (e.g. 5 for

¹ Dept. of Mathematics, University of California, Berkeley, CA 94720.

² Dept. of Mathematics and EECS Department, Division of Computer Science, University of California, Berkeley CA 94720.

the current implementation of LANSO) in addition to the demands of the operator A .

In exact arithmetic (c.f. [4]) all these methods produce the same approximate solution at each step. However they do differ in the way this approximate solution is computed. Since in practice the actual implementations of CG, SYMMLQ, and LAN are strongly affected by roundoff errors, the algorithms have quite different properties. Because of roundoff certain vectors which are orthogonal in exact arithmetic, cease to be so in finite precision arithmetic. But this loss of orthogonality does not prevent the convergence of the mentioned methods, it merely delays it. If however, the cost of a matrix vector multiplication dominates all other costs, then there is strong incentive to maintain a certain level of orthogonality and thus to keep the total number of calls on the operator A to a minimum.

In the case of the Lanczos algorithm this goal is achieved through the technique of selective orthogonalization (SO), which was developed by Parlett and Scott [7] for the eigenvalue problem. It amounts in storing the Lanczos vectors at each step and maintaining orthogonality among them. The Lanczos vectors are held in secondary storage. In order to maintain orthogonality and also to assemble the final solution they have to be recalled, in sequence, from time to time. Therefore one has also to include the cost of I/O operations in order to get an overall picture of the cost effectiveness of LANSO.

Another advantage of LANSO versus CG or SYMMLQ is that it can deal effectively with both definite and indefinite systems. A is called indefinite if it has positive and negative eigenvalues.

The implementation of the Lanczos algorithm with selective orthogonalization documented in this report closely follows the algorithm proposed by Parlett [6]. There is a certain overlap between this report and the paper by Parlett. A user, who wants to know more details about the Lanczos algorithm and SO is referred to the book [5] by Parlett. For a user, who wants to use the program LANSO only as a black box, it is sufficient to read section 3.1. and to consider the sample run in the appendix.

The notation will be along the following conventions: small Greek letters for scalars, small Roman letters for column vectors, capital Roman letters for matrices.

2. The Lanczos Algorithm with Selective Orthogonalization for Solving Systems of Linear Equations.

This section will provide a brief introduction into the theory behind LANSO. Most of the computational details are omitted here and delegated to section 3.

2.1. The Lanczos Algorithm in Exact Arithmetic.

Generally speaking at the j -th step of the Lanczos algorithm for solving (1.1) has computed an orthogonal basis for a certain j -dimensional subspace of R^n and the projection of A onto this subspace. The approximate solution to (1.1) at the j -th step is then the unique vector x_j in the j -dimensional subspace whose residual $b - Ax_j$ is orthogonal to the subspace. At the $j+1$ -st step a new vector is added to the orthogonal basis. In this way the approximate solution is found from a sequence of subspaces of increasing dimension. At the latest, when $j=n$, $b - Ax_n$ must be zero in theory and x_n is therefore the actual solution of (1.1).

The subspace under consideration is the Krylov subspace K^j of R^n defined by

$$K^j = \text{span} (b, Ab, A^2b, \dots, A^{j-1}b) . \quad (2.1.1)$$

Results of Kaniel [2] show that in important cases the residual norm $\|b - Ax_j\|$ becomes negligible for values of j much smaller than n . The Lanczos algorithm is simply the application of Gram-Schmidt orthogonalization to the vectors b, Ab, A^2b, \dots . On the first glance this appears to be a lot of work, but there are two facts, which make life easy and Lanczos worth doing.

Suppose at the j -th step an orthogonal basis of Lanczos vectors q_1, q_2, \dots, q_j for K^j has been computed. The q_j 's form the columns of the $n \times j$ matrix $Q_j = (q_1 \cdots q_j)$. Then the next task would be to orthogonalize $A^j b$ against $q_1 \cdots q_j$. Now some short considerations (c.f. Parlett[5]) show that it is only necessary to orthogonalize Aq_j against $q_1 \cdots q_j$. Furthermore Aq_j happens to be already orthogonal to q_1, q_2, \dots, q_{j-2} . So Aq_j has only to be orthogonalized against q_{j-1} and q_j using

$$r_j = Aq_j - q_j \alpha_j - q_{j-1} \beta_j \quad (2.1.2)$$

where $\alpha_j = q_j^* Aq_j$, $\beta_j = q_{j-1}^* Aq_j$. Note that q_1, q_2, \dots, q_{j-2} are not needed in (2.1.2). The core of a Lanczos step is therefore surprisingly simple. r_j still has to be normalized to become q_{j+1} . It turns out that $\|r_j\| = \beta_{j+1}$.

The special choice of the Krylov subspace K^j still gives another reward. T_j , the projection of A onto K^j in terms of Q_j is a tridiagonal matrix:

$$T_j \equiv Q_j^* A Q_j = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \beta_{j-1} & \alpha_{j-1} & \beta_j \\ \cdot & \cdot & 0 & \beta_j & \alpha_j \end{bmatrix} \quad (2.1.3)$$

The key relationships between the quantities computed by the Lanczos algorithm in exact arithmetic can be summarized in the three equations:

$$I_j - Q_j^* Q_j = 0 \quad (2.1.4)$$

$$A Q_j - Q_j T_j = r_j e_j^* \quad (2.1.5)$$

$$Q_j^* r_j = 0 \quad (2.1.6)$$

Here e_j (or $e_j^{(j)}$) is the j -th column of the $j \times j$ identity matrix I_j . The algorithm is started by setting $q_0 \equiv 0$ and $r_0 \equiv b$. It should be noted that the actual computation of the quantities q_j, α_j, β_j is done in a different order as in the above representation (c.f. section 3.3).

The next question is of course how to get the approximate solution vector x_j from the Krylov subspace K^j . In terms of Q_j and T_j , x_j can be characterized as follows

$$x_j \equiv Q_j T_j^{-1} Q_j^* b \quad (2.1.7)$$

Equation (2.1.7) says that x_j can be obtained by projecting the right hand side b onto K^j , then solving in K^j the system $T_j f_j = Q_j^* b$ where T_j is the projection of A , and finally expressing f_j in terms of the original space by forming $x_j = Q_j f_j$. Note that $Q_j^* b = e_1^{(j)} \beta_1$, since $q_1 = b / \beta_1$.

It is not clear that x_j defined by (2.1.7) is identical to the approximate solution defined at the beginning of this section, i.e. that $Ax_j - b$ is orthogonal to K^j . Using (2.1.4), (2.1.5), and (2.1.7) it follows that

$$\begin{aligned} Ax_j - b &= A Q_j T_j^{-1} Q_j^* b - b \\ &= (Q_j T_j + r_j e_j^*) T_j^{-1} Q_j^* b - b \\ &= r_j e_j^* T_j^{-1} Q_j^* b \\ &= r_j e_j^* T_j^{-1} e_1^{(j)} \beta_1 \end{aligned} \quad (2.1.8)$$

$$= \tau_j \varphi_j$$

where φ_j denotes the j -th component of the vector f_j . Thus the residual $Ax_j - b$ is a scalar multiple of the vector τ_j and by (2.1.6) indeed orthogonal to K_j .

By taking norms in (2.1.8) another key formula is obtained:

$$||Ax_j - b|| = ||\tau_j \varphi_j|| = \beta_{j+1} |\varphi_j| \quad (2.1.9)$$

This shows that the norm of the j -th residual can be found without forming explicitly x_j or the residual. It is not even necessary to compute f_j at each step in order to find φ_j . The details are in section 3.5.

Contrary to CG and SYMMLQ it is not necessary to update an approximate solution vector at each step. Only the scalar $\rho_j \equiv \beta_{j+1} |\varphi_j|$ has to be monitored. Only when some ρ_j becomes sufficiently small does the $j \times j$ system $T_j f_j = e^{(j)} \beta_1$ have to be solved. At this point it should be mentioned that if A is indefinite the T_j may have negative eigenvalues and we must take care in solving this system. The details are explained in section 3.7.

Finally the approximate solution x_j has to be computed. It is only here that the Lanczos vectors q_1, q_2, \dots, q_{j-2} are needed since x_j has to be assembled by forming $x_j = Q_j f_j$. In the present implementation the Lanczos vectors are written into secondary storage as soon as they are computed and, at the end, recalled one by one in order to form x_j .

2.2. Selective Orthogonalization.

A detailed account of the behaviour of the Lanczos algorithm in the presence of roundoff and of selective orthogonalization (SO) is available in Parlett [5]. Therefore in this section only the basic facts about SO will be presented to explain LANSO.

From now on let α_j, β_j, q_j etc. denote the corresponding quantities as they are computed, not their ideal counterparts. Let ε be the roundoff unit. The basic equations (2.1.4)-(2.1.6) are now perturbed by roundoff. Although for each j (2.1.5) is only slightly perturbed, the relations (2.1.4) and (2.1.6) completely fail after a certain number of steps depending on ε and on A . The Lanczos vectors, which are orthogonal in exact arithmetic, not only lose their orthogonality, but even become linearly dependent.

For a long time, as a remedy against this loss of orthogonality, it was suggested to reorthogonalize each new q_j against all previous Lanczos vectors,

which is of course very expensive. The results of Paige [3] give some better insight in the way how orthogonality is lost, and provide the theoretical basis for SO.

In order to explain Paige's results it is necessary to introduce certain quantities which are not of direct interest when solving $Ax = b$. Let $v_i^{(j)}$ be the eigenvalues and $s_i^{(j)}$ be the corresponding normalized eigenvectors of T_j

$$T_j s_i^{(j)} = s_i^{(j)} v_i^{(j)} \quad i = 1, \dots, j \quad (2.2.1)$$

From these one can compute the Ritz vectors $y_i^{(j)}$ by

$$y_i^{(j)} = Q_j s_i^{(j)} \quad (2.2.2)$$

These quantities change at each Lanczos step. If there is no confusion possible the superscripts will be dropped. The pairs $(v_i^{(j)}, y_i^{(j)})$ $i=1, \dots, j$ are approximate eigenpairs of A . The quality of this approximation can be determined by considering the residual $\|Ay_i - y_i v_i\|$. Applying (2.2.2) one finds that, to within roundoff,

$$\|Ay_i - y_i v_i\| \leq \beta_{ji} + \|F_j\| \quad (2.2.3)$$

where $\beta_{ji} \equiv \beta_{j+1} s_{ji}$ and $s_{ji} \equiv e_j^* s_i$, i.e., the bottom element of the corresponding eigenvector of T_j . The β_{ji} play an important role in the process of SO.

Now everything is ready to state one of the most important consequences of Paige's work, which can be summarized as follows: loss of orthogonality among the Lanczos vectors is equivalent to the convergence of a Ritz pair. In other words, if one of the β_{ji} becomes small, the corresponding Ritz pair converges to an eigenpair of A and the Lanczos vector q_{j+1} loses its orthogonality to q_1, \dots, q_{j-2} . But more is known about how q_{j+1} behaves, it is tilted towards $y_i^{(j)}$ while it is retaining its previous level of orthogonality to all other $y_k^{(j)}, k \neq i$. In order to maintain a certain level of orthogonality it is therefore only necessary to orthogonalize the new q_{j+1} , or equivalently the unnormalized τ_j against $y_i^{(j)}$ when the β_{ji} become smaller than some threshold. So first compute τ'_j (compare 2.1.2) by

$$\tau'_j \equiv Aq_j - q_j \alpha_j - q_{j-1} \beta_j \quad (2.2.4)$$

as usual, then check if any β_{ji} is small. If so then compute the corresponding $y_i^{(j)}$ and orthogonalize τ'_j against it obtaining

$$\tau_j \equiv \tau'_j - y_i^{(j)} \xi_i^{(j)} \quad (2.2.5)$$

where

$$\xi_i^{(j)} \equiv y_i^{(j)*} r_j / ||y_i^{(j)}||^2 \quad (2.2.6)$$

This requires the formation of $y_i = Q_j s_i$. The payoff for the expensive calculation comes later when subsequent Lanczos vectors, say q_{j+15} and q_{j+30} need to be orthogonalized against y_i , since a certain number m of Ritz vectors are also put in secondary storage and need not to be formed again. The question of how many Ritz vectors to keep, and whether to keep them in core in case the optimal number is small is still under investigation.

Finally it should be mentioned that the threshold for which a β_{ji} is considered to be small is set to be $\sqrt{\epsilon} ||T_j||$ in the present version of LANSO. This choice was based on the computational experience for the eigenvalue problem [7]. An analysis in [5] shows that this guarantees a certain level of orthogonality among the Lanczos vectors. The remaining computational details are discussed in section 3.6.

3. The Linear Equation Solver LANSO

The linear equation solver LANSO was developed and tested on the DEC VAX 11/780 of the Computer Science Division of the EECS Department at the University of California at Berkeley. It is coded in FORTRAN. There exists only a double precision version of the program, but it should be no problem to convert it to single precision. Throughout this section all FORTRAN variables are assumed to be of type DOUBLE PRECISION unless otherwise mentioned.

3.1. Usage of LANSO - The Subroutine LANCDR

A prospective user who is not interested in further details has to do only two things in order to compute an approximate solution vector for a system of linear equations:

- * provide a subroutine MATMUL which accomplishes the matrix vector multiplication,
- * write a main program which calls the subroutine LANCDR

An example is given in the appendix. The subroutine MATMUL has to be of the form

```
SUBROUTINE MATMUL(U,V,N)
  DIMENSION U(N),V(N)
```

Upon calling this subroutine, MATMUL has to compute $v + Au$ and write the result in v . This somewhat unusual form saves one n -vector storage (cf. section 3.3). There are no restrictions on the way Au is formed, which is one of the advantages of the Lanczos algorithm.

The header for LANCND is

```
SUBROUTINE LANCND (B,X,WORK,FAC,ISPC,N,M,LMAX,NTQ,NTR,IRES)
```

- B** is an array of dimension N , which contains the right hand side on input. The value of B is changed on output.
- X** is an array of dimension N , which contains an initial guess for the solution. If no initial guess is known, X has to be set to zero. On output X contains the approximate solution vector.
- WORK** is an array of dimension $ISPC$, providing the necessary workspace.
- FAC** is a number between 0 and 1. It denotes the factor by which the user wants to reduce the residual norm. After successful computation the solution vector in X satisfies $\|Ax - b\| \leq FAC \|b\|$. A word of caution on the use of FAC . If for example LANCND is called with $FAC = 10^{-4}$ and successfully computes a solution vector in X , then this does not mean that the result in X is correct to four digits. If for example the condition number of A is 10^5 the computed solution might not have a single correct digit.
- ISPC** is an integer, which denotes the dimension of the workarray $WORK$. At least 3 n -vectors and 10 j -vectors workspace are needed. Therefore $ISPC$ has to be larger than $3N + 10LMAX + 1$
- N** is an integer, the dimension of the matrix A .
- M** is an integer which roughly speaking, controls the amount of SO. (It is the number of Ritz vectors kept in secondary storage.) If one sets $M=0$ no SO will take place. The numerical experience so far has shown that $M=6$ or $M=4$ is a reasonable choice. M should be chosen larger, if it is known that the matrix A has several (more than 6) well separated eigenvalues at the end of its spectrum, or if a previous run of LANSO did not produce a good solution after a reasonable amount of time. The role of M

is still being investigated. In a final version of the program it will be set automatically.

LMAX is an integer which denotes the maximum number of Lanczos steps a user is willing to take.

NTQ, NTR, are integers which denote the channel number of the I/O unit for Lanczos vectors(**NTQ**) and Ritz vectors(**NTR**). It might be necessary to adjust the I/O operations to the local system.

IRES is an integer. If **IRES**=0 the required reduction of the residual norm was achieved within **LMAX** Lanczos steps. The solution in **X** is accurate within the limitations mentioned above(see **FAC**). If **IRES**=1 the program exceeded **LMAX** Lanczos steps. **X** still contains an approximate solution vector.

Since the work on **LANSO** is not yet completed this parameter list will be simplified in future.

The subroutine **LANCDR** only partitions the workspace **WORK** and calls the subroutine **LANSOL**, which is the core of the program.

3.2. The Subroutine **LANSOL**

The subroutine **LANSOL** has the following structure (for the notation see chapter 2):

- | |
|---|
| <ol style="list-style-type: none">1. Initialization
 set control parameters2. Loop: for $J \leq LMAX$ do<ol style="list-style-type: none">2.1 Simple Lanczos step2.2 Analyze T_j2.3 Update residual norm ρ_j2.4 Selective orthogonalization2.5 Check: if $\rho_i \leq TOL$ go to 3.3. Solve $T_j f_j = e_1 \beta_1$4. Assemble solution: $x_j = Q_j f_j$ |
|---|

Presently LANSOL is in modular form. Almost all the above steps are separate subroutines. In a final version this concept may be dropped for efficiency.

In the initialization step the guess for the solution x_g is used as follows: the residual $r \equiv b - Ax_g$ is formed. Then only the system $Ax_c = r$ is solved for the correction x_c . The Lanczos algorithm is initialized by $q \leftarrow 0$ and $\beta_1 \leftarrow ||r||$. The vectors q and r correspond exactly to the q_j and r_j in section 2.1. The tolerance TOL is set to be

$$TOL = \max(FAC, \epsilon) ||b||$$

3.3. Simple Lanczos Step - The Subroutine LANSIM

The subroutine LANSIM performs a simple Lanczos step according to the description in section 2.1. The following algorithm is used at the j-th step:

r	$\leftarrow r / \beta_j$
q	$\leftarrow -q \beta_j$
q	$\leftarrow q + Ar$
	swap q and r
α_j	$\leftarrow q^* r$
r	$\leftarrow r - q \alpha_j$
β_{j+1}	$\leftarrow r $

This somewhat peculiar form has two advantages: It uses only two n-vectors, and it follows the form of the algorithm, which was recommended by Paige [3] as least susceptible for roundoff. At the end of a simple Lanczos step the newly computed Lanczos vector q is written into secondary storage.

3.4. Analyze T_j - The Subroutine ANALZT

The purpose of this subroutine is to compute the eigenvalues of T_j and the corresponding β_{ji} . Presently a straight forward approach is used, i.e., at each step a subroutine TRIDQL is called, which performs the QL-algorithm for T_j and computes all the eigenvalues of T_j . Then all the β_{ji} are computed via the corresponding eigenvectors using a simple recurrence. This is far from optimal and an improved version of ANALZT is almost completed. This new version will use a more sophisticated updating technique, where only the α_i and β_{ji} are computed, which are of interest for SO.

3.5. Updating the residual norm ρ_j - The Subroutine UPRES.

In order to find the residual norm $\rho_j = \beta_{j+1}|\varphi_j|$, where φ_j is the bottom element of the solution vector f_j of the tridiagonal system of equations $T_j f_j = e_1 \beta_1$, theoretically one would have to solve this linear system at each step. But as T_j differs from T_{j-1} only by the last row and column this is not necessary.

In UPRES at the j -th step the matrix T_j is implicitly premultiplied by a scaled rotation matrix (cf. [5], pg.100), designed to make the element in position of the new β_j zero. Thus T_j is reduced to upper triangular form, but only implicitly. Only the effect of this operation on the last diagonal element and on the bottom element of the right hand side is stored in the global variables DELTA and GAMMA. Their ratio then yields φ_j , and DELTA and GAMMA as well as TT provide the necessary information for continuing the implicit reduction at the next step. The scaled rotations were used, since for indefinite A an almost singular T_j may occur and an ordinary LL^T decomposition of T_j might either brake down or produce erroneous results.

3.6. Selective Orthogonalization - The Subroutine SELORT

SELORT will be described here on a general level, since the intrinsic details would require more theoretical background than section 2.2 provides. There is a basic distinction between Ritz values which have already converged and those which have not, but might do so within the next few Lanczos steps. The former are called good Ritz values, the latter are called bad Ritz values. The good Ritz values are stored in GRITV, all Ritz values good and bad are also stored in RITVAL. Initially, when no Ritz value has converged yet, GRITV is empty.

The first step in SELORT is to check whether any of the good Ritz values has produced a small β_{ji} at the present Lanczos step. This is done by updating the arrays TAU and OLDTAU according to a recursion formula in [6]. If any of the TAU's gets small, this indicates a small β_{ji} and a SO has to be performed according to (2.2.5) and (2.2.6). Such an SO is called of the first type.

The second step in SELORT is to check certain of the bad Ritz values for a small β_{ji} . Which ones are to be checked is determined by the integers ICL and ICR. The mechanism behind ICL and ICR is explained in [5] and [6]. If any of the bad Ritz values really has converged, i.e., has a small β_{ji} , a SO will be performed. Then the converged Ritz value which has become good, will be added to the good Ritz values in GRITV and ICL and ICR are updated.

For each SO of the second type a new Ritz vector has to be computed. These Ritz vectors are written into secondary storage and from then on associated with the good Ritz values. Therefore these Ritz vectors are already available if a SO of the first type has to be performed, and do not have to be recomputed. The maximum number of good Ritz values and vectors to be kept is determined by the parameter M. After M Ritz vectors have been computed no more SO of the second type takes place. SO's of the first type will be still performed.

The idea behind this mechanism is that certain Ritz values, usually corresponding to the extreme eigenvalues of A , will force SO's very often. These are also the ones which will converge first. It is therefore often necessary to orthogonalize against the corresponding Ritz vectors and their expensive recomputation has to be avoided.

Finally it should be mentioned that each SO changes the elements of the matrix T_j slightly. How SO affects the structure of T_j is again explained in [6]. These changes and their possible effects on ρ_j and the solution vector f_j are recorded in CHALF, CHBET, and CHF by calling the subroutine UPDATE.

3.7. Solving $T_j f_j = e_1 \beta_1$ - The subroutine TRISOL

The subroutine TRISOL is a general purpose subroutine for solving ill conditioned symmetric tridiagonal systems. As for UPRES, fast scaled rotations are used. The subroutine has the additional feature that the elements of T_j are unchanged. All information (the cotan of the angles of rotation) is stored in the array COT.

References

- [1] M.R. Hestenes and E.Stiefel, Methods of Conjugate Gradients for Solving Linear Systems, J.Res.Bur.Standards, 49, 409-436 (1952).
- [2] S. Kaniel, Estimates for some Computational Techniques in Linear Algebra, Math. Comp. 20,369-378 (1966).
- [3] C.C. Paige, Computational Variants of the Lanczos Method for the Eigenproblem, J.Inst.Math.Appl. 10,373-381 (1972).
- [4] C.C. Paige and M.A.Saunders, Solution of sparse indefinite systems of linear equations, SIAM J.Num.Anal. 12, 617-629 (1975).
- [5] B.N. Parlett, The Symmetric Eigenvalue Problem , Prentice-Hall, Englewood Cliffs (1980).
- [6] B.N. Parlett, A New Look at the Lanczos Algorithm for Solving Symmetric Systems of Linear Equations, Lin.Alg.Appl.29, 323-346 (1980).
- [7] B.N. Parlett and D.Scott, The Lanczos Algorithm with Selective Orthogonalization, Math.Comp.,33, 217-238, (1979).

Appendix

As a first example consider the matrix $A = (a_{ij})$ where

$$a_{ij} = \begin{cases} 4 & \text{if } i = j \\ -1 & \text{if } i = j+1, i = j-1, i = j+15, i = j-15 \\ 0 & \text{otherwise} \end{cases}$$

The correct solution is $x = (1,1,\dots,1)^T$, the right hand side vector b is generated by calling `matmul` once. The dimension $n=100$, and the other input parameters can be directly read off from the following program listing:

```
1      program main
2      implicit double precision (a-h,o-z)
3      dimension b(100),x(100)
4      dimension work(1100)
5      c..
6      c.. initializing data for lancdr
7      c..
8      n=100
9      lmax=60
10     ntq=8
11     ntr=9
12     m=8
13     ispc=1100
14     fac=1.0d-8
15     do 10 i=1,n
16         x(i)=1.0d0
17         b(i)=0.0d0
18 10    continue
19     call matmul(x,b,n)
20     do 11 i=1,n
21         x(i)=0.0d0
22 11    continue
23     write (6,9000)
24     write (6,9010) n
25     write (6,9011) lmax
26     write (6,9012) fac
27     write (6,9013) m
28     call lancdr(b,x,work,fac,ispc,n,m,lmax,ntq,ntr,ires)
29     if (ires.eq.0) write (6,9005)
30     if (ires.eq.1) write (6,9007)
31     write (6,9004) (x(i), i=1,n)
32     c..
33     c.. computing relative error
34     c..
35     err=0.0d0
36     do 20 i=1,n
37 20     err=err+(1.0d0-x(i))**2
38     err=dsqrt(err)/100.d0
39     write (6,9020) err
40     stop
```

```

41 9000 format (/,3x," Solving Ax=b with the Lanczos algorithm ",//)
42 9004 format ( d26.18 )
43 9005 format (/, " residual norm became small, solution vector = ",//)
44 9007 format (/, " maximum number of Lanczos steps exceeded, approx. solu
45      tion vector = ",//)
46 9010 format (/, " Number of Equations = ",i5,/ )
47 9011 format (/, " Max. Number of Lanczos Steps = ",i5,/ )
48 9012 format (/, " Factor for reducing res.norm = ",d12.4,/ )
49 9013 format (/, " Max. Number of Ritzvectors kept on tape = ",i4,/ )
50 9020 format (/, " Rel. Error in the solution = ",d12.4,/ )
51      end
52

```

```

53      subroutine matmul (u,v,n)
54      implicit double precision (a-h,o-z)
55      dimension u(n),v(n)
56      nm1=n-1
57      v(1)=v(1)+4.0d0*u(1)-u(2)
58      do 10 i=2,nm1
59          v(i)=v(i)+4.0d0*u(i)-u(i-1)-u(i+1)
60 10  continue
61      v(n)=v(n)+4.0d0*u(n)-u(nm1)
62      nm15=n-15
63      do 11 i=1,nm15
64          v(i)=v(i)-u(i+15)
65          v(i+15)=v(i+15)-u(i)
66 11  continue
67      return
68      end

```

The subroutine matmul shows how the special form of this matrix can be exploited. For this example no additional storage for the matrix A is necessary. Here LANSO computes the approximate solution in 25 steps without performing an selective orthogonalization.

As a second example a matrix M is considered, which is obtained from the diagonal matrix $\Lambda = \text{diag} (\lambda_1, \lambda_2, \dots, \lambda_{100})$ by a similarity transformation of the form $M = P\Lambda P$. Here P is the reflector $I - 2\gamma w w^*$, where the n - vector w was chosen at random. M has the eigenvalues

$$\lambda_i = \begin{cases} 10[i/3]+1, & i = 1, \dots, 90 \\ 1900 + i, & i = 91, \dots, 100 \end{cases}$$

where $[i/3]$ is the largest integer not exceeding $\frac{i}{3}$. The matrix M is actually a full matrix, but the special choice of the reflector P makes it possible to store it in implicit form in only three arrays of dimension(100), (c.f. [5, pg. 119]). Although the matrix M is never explicitly computed, LANSO can solve $Mx = b$. As before the solution vector is $x = (1, \dots, 1)^*$, and the right hand side is computed by calling matmul. The other parameters can be found directly in the program listing:

```
1      program main
2      implicit double precision (a-h,o-z)
3      dimension b(100),x(100)
4      dimension work(1100)
5      common /mat/ d(100),w(100),p(100)
6      c..
7      c..  setting up data for matrix
8      c..
9      do 1 i=1,90
10 1      d(i)=10.0d0*dfloat(i/3)+1.0d0
11      do 2 i=91,100
12 2      d(i)=1900.0d0+dfloat(i)
13      do 3 i=1,100
14 3      w(i)=runi(1.0d0)
15      gamma=2.0d0/dot(w,w,100)
16      do 4 i=1,100
17 4      p(i)=gamma*d(i)*w(i)
18      delta=0.5d0*gamma*dot(w,p,100)
19      do 5 i=1,100
20 5      p(i)=p(i)-delta*w(i)
21      c..
22      c..  initializing data for lancdr
23      c..
24      n=100
25      lmax=60
26      ntq=8
27      ntr=9
28      m=6
29      ispc=1100
30      fac=1.0d-8
31      do 10 i=1,n
32          x(i)=1.0d0
33          b(i)=0.0d0
34 10      continue
35      call matmul(x,b,n)
36      do 11 i=1,n
37          x(i)=0.0d0
38 11      continue
39      write (6,9000)
40      write (6,9010) n
41      write (6,9011) lmax
42      write (6,9012) fac
43      write (6,9013) m
44      call lancdr(b,x,work,fac,ispc,n,m,lmax,ntq,ntr,ires)
45      if (ires.eq.0) write (6,9005)
46      if (ires.eq.1) write (6,9007)
47      write (6,9004) (x(i), i=1,n)
48      c..
49      c..  computing relative error
50      c..
51      err=0.0d0
52      do 20 i=1,n
53 20      err=err+(1.0d0-x(i))**2
54      err=dsqrt(err)/100.d0
```

```
55     write (8,9020) err
56     stop
57 9000 format (/,3x," Solving Ax=b with the Lanczos algorithm ",//)
58 9004 format ( d26.18 )
59 9005 format (/, " residual norm became small, solution vector = ",// )
60 9007 format (/, " maximum number of Lanczos steps exceeded, approx. solu
61     tion vector = ",// )
62 9010 format (/, " Number of Equations = ",i5,/ )
63 9011 format (/, " Max. Number of Lanczos Steps = ",i5,/ )
64 9012 format (/, " Factor for reducing res.norm = ",d12.4,/ )
65 9013 format (/, " Max. Number of Ritzvectors kept on tape = ",i4,/ )
66 9020 format (/, " Rel. Error in the solution = ",d12.4,/ )
67     end
68

69     subroutine matmul (u,v,n)
70     implicit double precision (a-h,o-z)
71     dimension u(n),v(n)
72     common /mat/ d(100),w(100),p(100)
73     temp1=dot(u,p,n)
74     temp2=dot(u,w,n)
75     do 10 i=1,n
76 10     v(i)=v(i)+d(i)*u(i)-temp1*w(i)-temp2*p(i)
77     return
78     end
```

LANSO takes 37 steps to find the solution to the required accuracy.

OUTPUT FOR EXAMPLE 1
=====

Solving $Ax=b$ with the Lanczos algorithm

Number of Equations = 100

Max. Number of Lanczos Steps = 60

Factor for reducing res.norm = .1000e-05

Max. Number of Ritzvectors kept on tape = 6

residual norm became small, solution vector =

.999999637083817405e+00
.999999759774071451e+00
.999999826580549289e+00
.100000020016811733e+01
.100000014506565835e+01
.100000025057870801e+01
.999999833306143926e+00
.100000008249720648e+01
.999999848481511511e+00
.100000005159510005e+01
.100000018989151521e+01
.100000014946928509e+01
.100000038902233354e+01
.100000005965066255e+01
.999999939745708244e+00
.999999659002722496e+00
.999999718438703877e+00
.100000009394325035e+01
.100000029142185756e+01
.100000032042909026e+01
.100000028993623713e+01
.999999834648724728e+00
.999999752075789194e+00
.999999483226686059e+00
.999999554435604909e+00
.999999733085768083e+00
.100000000687028023e+01
.100000048370139111e+01
.100000037029570341e+01
.100000018169424595e+01
.100000001875801156e+01
.9999993839039896088e+00
.100000005367251879e+01
.100000019650787603e+01
.100000035297968679e+01

.100000013969025162e+01
.100000012121468090e+01
.999999740722122210e+00
.999999692331521484e+00
.999999633507842511e+00
.999999725918109061e+00
.100000030600526440e+01
.10000003589758790e+01
.100000053564985153e+01
.100000022745163697e+01
.999999895186535470e+00
.999999307520274864e+00
.999999716084310730e+00
.999999979901299985e+00
.100000014574085189e+01
.100000014574085186e+01
.99999979901300073e+00
.999999716084310819e+00
.999999307520274908e+00
.99999935186535492e+00
.100000022745163702e+01
.10000003561985156e+01
.100000003889788796e+01
.10000000600526437e+01
.999999725918109061e+00
.999999633507842467e+00
.999999692331521484e+00
.999999740722122232e+00
.100000012121468085e+01
.100000013969025159e+01
.100000035297968679e+01
.100000019650787608e+01
.100000035367251890e+01
.999999839039896110e+00
.10000001875801164e+01
.100000018169424601e+01
.100000037029570335e+01
.100000048370139111e+01
.100000000687028029e+01
.999999733085768128e+00
.999999554435604865e+00
.999999483226686037e+00
.999999752075789194e+00
.999999834648724728e+00
.100000028993623713e+01
.100000032042909026e+01
.100000029142185756e+01
.100000009394325040e+01
.999999718438703900e+00
.999999659002722541e+00
.999999939745708266e+00
.100000005965066258e+01
.100000038902233360e+01
.100000014946928506e+01
.100000018989151521e+01
.100000005159510000e+01
.999999848481511533e+00
.100000008249720648e+01
.999999833306143926e+00

.100000025057870801e+01
.100000014506565840e+01
.100000020016811730e+01
.999999826580549378e+00
.999999759774071473e+00
.999999637083817428e+00

Rel. Error in the solution = .2536e-07

OUTPUT FOR EXAMPLE 2 =====

Solving $Ax=b$ with the Lanczos algorithm

Number of Equations = 100

Max. Number of Lanczos Steps = 60

Factor for reducing res.norm = .1000e-05

Max. Number of Ritzvectors kept on tape = 6

residual norm became small, solution vector =

.999999863927111488e+00
.999999863920964716e+00
1.00000001900409702e+01
1.00000001899048524e+01
1.00000001897568491e+01
.999999936922472821e+00
.999999936909260279e+00
.999999936894842612e+00
1.00000021974542860e+01
1.00000021970864120e+01
1.00000021966832375e+01
.999999325553815988e+00
.999999325586910426e+00
.999999325623372726e+00
1.00000174537373560e+01
1.00000174512040407e+01
1.00000174483949719e+01
.999996234638208925e+00
.999996235222299767e+00
.999996235875080974e+00
1.00000669363873743e+01
1.00000669196794215e+01
1.00000669008239182e+01
.999990352898234169e+00
.999990356137667091e+00
.999990359838556753e+00
1.00001086069744610e+01
1.00001085482205890e+01
1.00001084800300200e+01
.999991303205643423e+00
.999991310247189924e+00
.999991318592492884e+00
1.00000324247106470e+01
1.0000032373509389e+01

1.00000323110496561e+01
1.00000288302206905e+01
1.00000287429263651e+01
1.00000286318340945e+01
.999993926757515328e+00
.999993961558802291e+00
.999994008934411527e+00
1.00000448197019129e+01
1.00000439871208763e+01
1.00000427054673532e+01
1.00000069214497211e+01
1.00000060490449402e+01
1.00000042649670412e+01
.999999419382194099e+00
1.00000060774708513e+01
1.00000264152592533e+01
.999990822092653575e+00
1.00000149794274357e+01
1.00000510369343623e+01
.999999212948167959e+00
.999999161492185107e+00
.999999132305229543e+00
.999996754408291488e+00
.999996672530961050e+00
.999996629364000467e+00
1.00000512532776342e+01
1.00000516668752543e+01
1.00000519852928779e+01
.999995196358234439e+00
.999995180051144805e+00
.999995166759212073e+00
1.00000326389192168e+01
1.00000327114813153e+01
1.00000327728637425e+01
.999998267002337582e+00
.999998265217753546e+00
.99999826368313037e+00
1.00000071694769677e+01
1.00000071780890909e+01
1.00000071857116762e+01
.999999760511505634e+00
.999999760587486586e+00
.999999760626095435e+00
1.00000005782013424e+01
1.00000005797964558e+01
1.00000005812461823e+01
.999999986461361767e+00
.999999986558888954e+00
.999999986648372508e+00
.999999999807182616e+00
.999999999836449453e+00
.999999999959751440e+00
.999999998334980167e+00
.999999998397753198e+00
.999999998456186412e+00
.99999999860640732e+00
.9999999987130161e+00
.999999994215248811e+00
1.00000000012736054e+01

.99999998435428750e+00
.99999998548833502e+00
1.0000000001488240e+01
.99999999699726216e+00
.999999996039172560e+00
.99999999005462814e+00
.99999999035764087e+00

Rel. Error in the solution = .4023e-06